34장 이터러블

34.1 이터레이션 프로토콜

- 이터레이션 프로토콜은 **순회 가능한 데이커 컬렉션(자료구조)을 만들기 위해** ECMAScript 사양에 정의하여 미리 약속한 규칙
- ES6에서는 순회 가능한 데이터 컬렉션을 이터레이션 프로토콜을 준수하는 이터러블로 통일하여 for...of문, 스프레드 문법, 배열 디스트럭처링 할당의 대상으로 사용할 수 있도록 일원화
- 이터레이션 프로토콜
  - **이터러블** 프로토콜
  - **이터레이터** 프로토콜

34.1.1 이터러블

Symbol.iterator를 프로퍼티 키로 사용한 메서드를 직접 구현하거나 프로토타입 체인을 통해 상속받은 객체

```
const isIterable = v => v !== null && typeof v[Symbol.iterator] === 'function';

// 배열, 문자열, Map, Set 등은 이터러블이다.
isIterable([]); // true
isIterable(''); // true
isIterable(new Map()); // true
isIterable(new Set()); // true
isIterable({}); // false

const arr = [1, 2, 3];

// 배열은 Array.prototype의 Symbol.iterator 메서드를 상속받는 이터러블이다
console.log(Symbol.iterator in arr); // true

// 이터러블인 배열은 for...of문으로 순회 가능하다.
```

```
for(const item of arr) {
  console.log(item);
}

// 이터러블인 배열은 스프레드 문법의 대상으로 사용할 수 있다.
console.log([...arr]); // [1, 2, 3]

// 이터러블인 배열은 배열 디스트럭처링 할당의 대상으로 사용할 수 있다.
const [a, ...rest] = arr;
console.log(a, rest); // 1 [2, 3]
```

일반 객체는 이터러블 프로토콜을 준수한 이터러블이 아니다.

```
const obj = { a: 1, b: 2 };

// 일반 객체는 Symbol.iterator 메서드를 구현하거나 상속받지 않는다.
console.log(Symbol.iterator in obj); // false

// 이터러블이 아닌 객체는 for...of문으로 순회할 수 없다.
for(const item of obj) {
  console.log(item); // TypeError
}

// 이터러블이 아닌 객체는 배열 디스트럭처링 할당의 대상으로 사용할 수 없다.
const [a, b] = obj; // TypeError

// 객체 리터럴 내부에서 스프레드 문법의 사용을 허용한다.
console.log({ ...obj }); // { a: 1, b: 2 }
```

34.1.2 이터레이터

- 이터러블의 Symbol.iterator 메서드를 호출하면 이터레이터 프로토콜을 준수한 이터레이터를 반환
- 이때 이터레이터는 next 메서드를 갖는다.

```
const arr = [1, 2, 3]; // 이터러블
```

```
const iterator = arr[Symbol.iterator](); //이터레이터를 반환

// next 메서드를 갖는다.
console.log('next' in iterator); // true
```

이터레이터의 next 메서드는 이터러블의 각 요소를 순회하기 위한 포인터 역할

```
const arr = [1, 2, 3];

const iterator = arr[Symbol.iterator](); // 이터레이터 반환

// next 메서드를 호출하면 이터러블을 순회하며 순회 결과를 나타내는 이터레이터 리절트 객체를 반환한다.
// 이터레이터 리절트 객체는 value와 done 프로퍼티를 갖는다.
console.log(iterator.next()); // { value: 1, done: false }
console.log(iterator.next()); // { value: 2, done: false }
console.log(iterator.next()); // { value: 3, done: false }
console.log(iterator.next()); // { value: undefined, done: true }
```

34.2 빌트인 이터러블

자바스크립트는 이터레이션 프로토콜을 준수한 객체인 빌트인 이터러블을 제공

빌트인 이터러블	Symbol.iterator 메서드
Array	Array.prototype[Symbol.iterator]
String	String.prototype[Symbol.iterator]
Map	Map.prototype[Symbol.iterator]
Set	Set.prototype[Symbol.iterator]
TypedArray	TypedArray.prototype[Symbol.iterator]
arguments	arguments[Symbol.iterator]
DOM 컬렉션	HTMLCollection.prototype[Symbol.iterator], NodeList.prototype[Symbol.iterator]

34.3 for...of 문

for...of 문은 이터러블을 순회하면서 이터러블의 요소를 변수에 할당

`for(변수선언문 of 이터러블) { ... }`

for...in문의 형식과 매우 유사하지만 for...in문은 프로퍼티 키가 심벌인 프로퍼티는 열거하지 않는다.

`for(변수선언문 in 객체) { ... }`

```
for(const item of [1, 2, 3]) {
  // item 변수에 순차적으로 1, 2, 3이 할당된다.
  console.log(item); // 1 2 3
}
```

34.4 이터러블과 유사 배열 객체

- 유사 배열 객체는 이터러블이 아닌 일반 객체
  - 따라서 Symbol.iterator 메서드가 없기 때문에 for...of 문으로 순회 X

```
// 유사 배열 객체
const arrayLike = {
  0: 1,
  1: 2,
  2: 3
}
// 유사 배열 객체는 순회할 수 없다.
for (const item of arrayLike) {
  console.log(item); // 1 2 3
} // -> TypeError
```

- arguments, NodeList, HTMLCollection은 유사 배열 객체이면서 이터러블
- 배열도 마찬가지로 이터러블이 도입되면서 Symbol.iterator 메서드를 구현하여 이터러블이 되었다.
- Array.from 메서드는 유사 배열 객체 또는 이터러블을 인수로 전달받아 배열로 변환하여 반환

```
const arrayLike = {
  0: 1,
  1: 2,
  2: 3
};

const arr = Array.from(arrayLike);
console.log(arr); // [1, 2, 3]
```

34.5 이터레이션 프로토콜의 필요성

이터레이션 프로토콜은 다양한 데이터 공급자(Array, String, Map/Set, DOM 컬렉션)가 하나의 순회 방식을 갖도록 규정하여 데이터 소비자(for...of, 스프레드 문법, 배열 디스트럭처링 할당, Map/Set 생성자)가 효율적으로 다양한 데이터 공급자를 사용할 수 있도록 데이터 소비자와 데이터 공급자를 연결하는 인터페이스의 역할을 한다.

34.6 사용자 정의 이터러블

34.6.1 사용자 정의 이터러블 구현

- 사용자 정의 이터러블은 이터레이션 프로토콜을 준수하도록 Symbol.iterator 메서드를 구현하고 Symbol.iterator 메서드가 next 메서드를 갖는 이터레이터를 반환
- next 메서드는 done과 value 프로퍼티를 가지는 이터레이터 리절트 객체를 반환
- for...of문은 done프로퍼티가 true가 될 때까지 반복하다가 true가 되면 반복을 중지

34.6.2 이터러블을 생성하는 함수

```
// 사용자 정의 이터러블을 반환하는 함수
// 수열의 최대값을 인수로 전달받음
const fibonacciFunc = function (max) {
  let [pre, cur] = [0, 1]; // 배열 디스트럭처링 할당

  // Symbol.iterator 메서드를 구현한 이터러블을 반환
  return {
    [Symbol.iterator]() {
      return {
```

```
        next() {
          [pre, cur] = [cur, pre + cur];
          return { value: cur, done: cur >= max };
        }
      };
    }
  };
};

for (const num of fibonacciFunc(10)) {
  console.log(num) // 1 2 3 5 8
}
```

34.6.3 이터러블이면서 이터레이터인 객체를 생성하는 함수

이터러블이면서 이터레이션인 객체를 생성하면 Symbol.iterator 메서드를 호출하지 않아도 된다.

```
// 이터러블이면서 이터레이터인 객체를 반환하는 함수
const fibonacciFunc = function (max) {
  let [pre, cur] = [0, 1]<; // 배열 디스트럭처링 할당

  // Symbol.iterator 메서드와 next 메서드를 소유한
  return {
    [Symbol.iterator]() { return this; },
    next() {
      [pre, cur] = [cur, pre + cur];
      return { value: cur, done: cur >= max }; // 리절트 객체
반환
    }
  };
};

let iter = fibonacciFunc(10) // 이터러블이면서 이터레이터

// 이터러블이므로 for of 문 순회가능
for (const num of fibonacciFunc(10)) {
  console.log(num) // 1 2 3 5 8
}
```

```
// 이터레이터이므로 이터레이션 리절트 객체를 반환하는 next 메서드 소유
console.log(iter.next()); // { value: 1, done: false }
console.log(iter.next()); // { value: 2, done: false }
...
console.log(iter.next()); // { value: 13, done: true }
```